Report No. 437

PARALLEL METHODS AND BOUNDS OF EVALUATING POLYNOMIALS

by

Kiyoshi Maruyama

March, 1971

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 437

PARALLEL METHODS AND BOUNDS OF EVALUATING POLYNOMIALS

by

Kiyoshi Maruyama

March, 1971

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the Department of Computer Science at the University of Illinois at Urbana-Champaign, Urbana, Illinois; the National Science Foundation under NSF Grant GJ-328.

Abstract --- a lower bound of the maximum evaluable degree of polynomia
each step s is given, and is $N^*(s) > 2^{s(1-\delta)}$, $\delta \approx (2/s)^{1/2}$. An upper b
is derived for the number of steps required to evaluate polynomials of
n, and is given by $T(P_n) < (1+\epsilon)\log_2 n$, which approaches theoretical l
bound $\lceil \log_2(2n+1) \rceil$ as $n \to \infty$, where $\epsilon \approx (2/\log_2 n)^{1/2}$.

An upper bound of the number of operations required to achiev
e polynomials of de
construction of comp
ods, which achieves
better results than
e of a simple schedu

primal problem, dua
, LOF, consistency,

CORRECTIONS

1. In Theorem 3 (primal property) on page 15 should be $(2/\log_2 n)^{1/2}$.

2. In Theorem 7 on page 32, the statement "The minimum number of ....." should be "The upper bound of number of .....".

   Better upper bound of number of operations required to evaluate polynomials of degree n by Muraoka's folding method is given by $2n + \eta - 2$, where η is an integer $\eta \geq 1$ such that η-th Fibonacci number $\leq n$ and $(\eta+1)$ st Fibonacci number $> n$. For example

η	: 1	2	3	4	5	6	7	8	...
Fibonacci No.	: 1	1	2	3	5	8	13	21	...

   if n=7 then $\eta = 5$ thus the number of operations required to evaluate a polynomial of degree 7 is $2 \cdot 7 + 5 - 2 = 17$, which is better than Theorem 7.

Abstract --- a lower bound of the maximum evaluable degree of polynomial for each step s is given, and is $N^*(s) > 2^{s(1-\delta)}$, $\delta \approx (2/s)^{1/2}$. An upper bound is derived for the number of steps required to evaluate polynomials of degree n, and is given by $T(P_n) < (1+\epsilon)\log_2 n$, which approaches theoretical lower bound $\lceil \log_2(2n+1) \rceil$ as $n \to \infty$, where $\epsilon \approx (2/\log_2 n)^{1/2}$.

An upper bound of the number of operations required to achieve the theoretical lower bound of steps required to evaluate polynomials of degree n is given as Cn, $C > 2$. Furthermore a systematic construction of computation trees, multi-folding and modified multi-folding methods, which achieves the bound is given. Our dual problem approach leads to better results than Brent's [1] primal problem approach, and shows the existance of a simple scheduling algorithm to evaluate polynomials within the bound.

Index terms --- evaluation, n-th degree polynomial, primal problem, dual problem, steps, computation tree, binary, balanced, LHT, RHT, LOF, consistency, segment, subpolynomial, x-term, amplification factor.

ACKNOWLEDGMENT

The author wishes to thank Dr. D. Kuck, Professor of Computer Science at University of Illinois, for suggesting the problem and his comments. He also acknowledges the assistance of Dr. Y. Muraoka and his helpful discussion. Thanks are extended to Miss Sue Cook who helped in getting this report finished.

TABLE OF CONTENTS

1. INTRODUCTION

The computation of polynomials has been studied for many years. It has been showm by Pan[5] that 2n operations are required to evaluate a polynomial of degree n. Thus, for a serial machine, Horner's rule is optimal. However, it is easy to see that it requires only $\lceil \log_2 n \rceil$ steps to evaluate all powers, one step to multiply by the coefficients, and $1 + \lceil \log_2 n \rceil$ steps to sum the terms. Thus by introducing some "redundant" operations one can obtain the result in $2(1 + \lceil \log_2 n \rceil)$ time steps(assuming multiplication and addition times are equal). This is a crude bound for a multiarithmetic unit machine because some additions can be performed before the final multiplications are performed. Some of the well known methods are Estrin's[3] and the k-th order Horner's rule[2]. And two other methods called a tree method and a folding method have been developed by Muraoka[4]. Hereafter, we assume that arbitrarily many processors are available to evaluate polynomials.

We write an n-th degree polynomial as

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots\ldots\ldots\ldots + a_1 x + a_0 \qquad (1)$$

where we assume $a_i \neq 0$ for $0 \leq i \leq n$. P_n may be used to denote $P_n(x)$ for simplicity. The lower bound for the number of steps, or time, required to evaluate a polynomial of degree n is $\lceil \log_2 (2n + 1) \rceil$, or $\lceil \log_2 n \rceil + 1$, since 2n operations are required to evaluate $P_n(x)$ while a k level balanced binary tree contains 2^{k-1} nodes which correspond to binary operators.

The dual problem to a problem of finding the minimum number of steps s required to evaluate $P_n(x)$ may be stated as a problem of finding the maximum N for which the polynomial $P_N(x)$ may be evaluated for a particular given method in s steps. We consider this dual problem hereafter.

Let $P_{n_1}(x)$ and $P_{n_2}(x)$ be two polynomials and let s_1 and s_2 be the minimum number of steps required to evaluate P_{n_1} and P_{n_2}, respectively. Clearly then $n_1 > n_2$ implies $s_1 \geq s_2$(the converse is not true) and $s_1 > s_2$ implies $n_1 > n_2$(see appendix B).

$P_n(x)$ may be divided into q segments where each segment consists of a subpolynomial and an x-term, i.e., $Q_{n_i}(x)$ and x^{m_i}. Thus we may write

$$P_n(x) = \sum_{i=1}^{q-1} Q_{n_i}(x)x^{m_i} + P_m(x) \qquad (2)$$

where $n = m + \sum_{i=1}^{q-1} (n_i + 1)$ and $m_i = m + i + \sum_{j<i} n_j$. Such a segmentatin of a polynomial is called a q-cut. To compute a segment in s' steps, both subpolynomial and x-term should be evaluated in (s' - 1) steps, i.e., $s' - 1 \geq T(Q_{n_i})$ and $s' - 1 \geq \lceil \log_2 m_i \rceil$, where T is a function from polynomials P_n to the non negative integers I. Such a segment is said to be consistent if both conditions are satisfied. Unless otherwise specified, for each $Q_{n_i}(x)$ we choose $n_i = N$ such that $N = \max\{ m \mid s - 1 = T(P_m)\}$.

The symbol N(s,q), where s is non negative integer and q is integer greater than or equal to 2, is used to denote the maximum degree of polynomial $P_{N(s)}$ which can be evaluated in s steps by q-cut. It is convenient to consider the formula (2) as a computation tree to evaluate $P_n(x)$. The sub computation tree to evaluate the first term of (2) is called the LHT while the sub computation tree for the second term is called the RHT.

As an example, Muraoka's folding method, q = 2, may be stated as follows: let N(s-1) and N(s) be the maximum degrees of polynomials which can be evaluated in (s - 1) and s steps, respectively. Then using his method, N(s+1) = N(s-1) + 1 + N(s), where N(s-1) + 1 and N(s) correspond to the LHT and RHT of the computation tree of $P_{N(s+1)}$, and N(s+1) approaches 1.62 times N(s) for relatively large s since the sequence $\{N(s) + 1 \mid s \geq 0\}$ forms a Fibonacci

sequence(reference Table 1).

2. COMPUTATION TREES

In general, for such a computation tree, if $2^k < q - 1 \leq 2^{k+1}$, $k \geq 0$, then the maximum degree of polynomial which can be evaluated in $(s + 1)$ steps, $N(s+1, q)$, is given by:

$$\begin{aligned} N(s+1,q) = N(s) &+ A_0(N(s-k-1+M) + 1) + A_1(N(s-k-2+M) + 1) + \cdots\cdots \\ &+ A_M(N(s-k-1) + 1) + A_{M+1}(N(s-k-2) + 1) + \cdots\cdots \\ &+ A_{M+L}(N(s-k-1-L) + 1) \\ = N(s) &+ \sum_{i=0}^{M+L} A_i(N(s-k-1+M-i) + 1) \end{aligned} \tag{3}$$

where $q - 1 = \sum_{i=0}^{M+L} A_i$, A_i is the number of segments on the level $(k-M+i)$ and A_{M+L} is multiple of 2. L is the level difference between k and the highest level of segment in the LHT. M is the level difference between k and the lowest level of segment in the LHT.

A computation tree is said to be q-consistent if each segment in the LHT is consistent. Thus formula (3) is valid if its computation tree is q-consistent, and is illustrated in Figure 1. It is easy to see that a computation tree is q-consistent if and only if the x-term of the leftmost segment in A_{M+L} is consistent.

A balanced computation tree with respect to q, $2^k < q - 1 \leq 2^{k+1}$, is a binary tree such that $M = 0$ and $0 \leq L \leq 1$ (or $q - 1 = A_0 + A_1$ and $A_i = 0$ for $i \neq 0, 1$). A balanced computation tree is said to be completely balanced if $M = 0$ and $L = 1$(or $q - 1 = 2^k = A_1$ and $A_i = 0$ for $i \neq 1$). A LOF(lower order first) computation tree is a binary balanced tree such that the

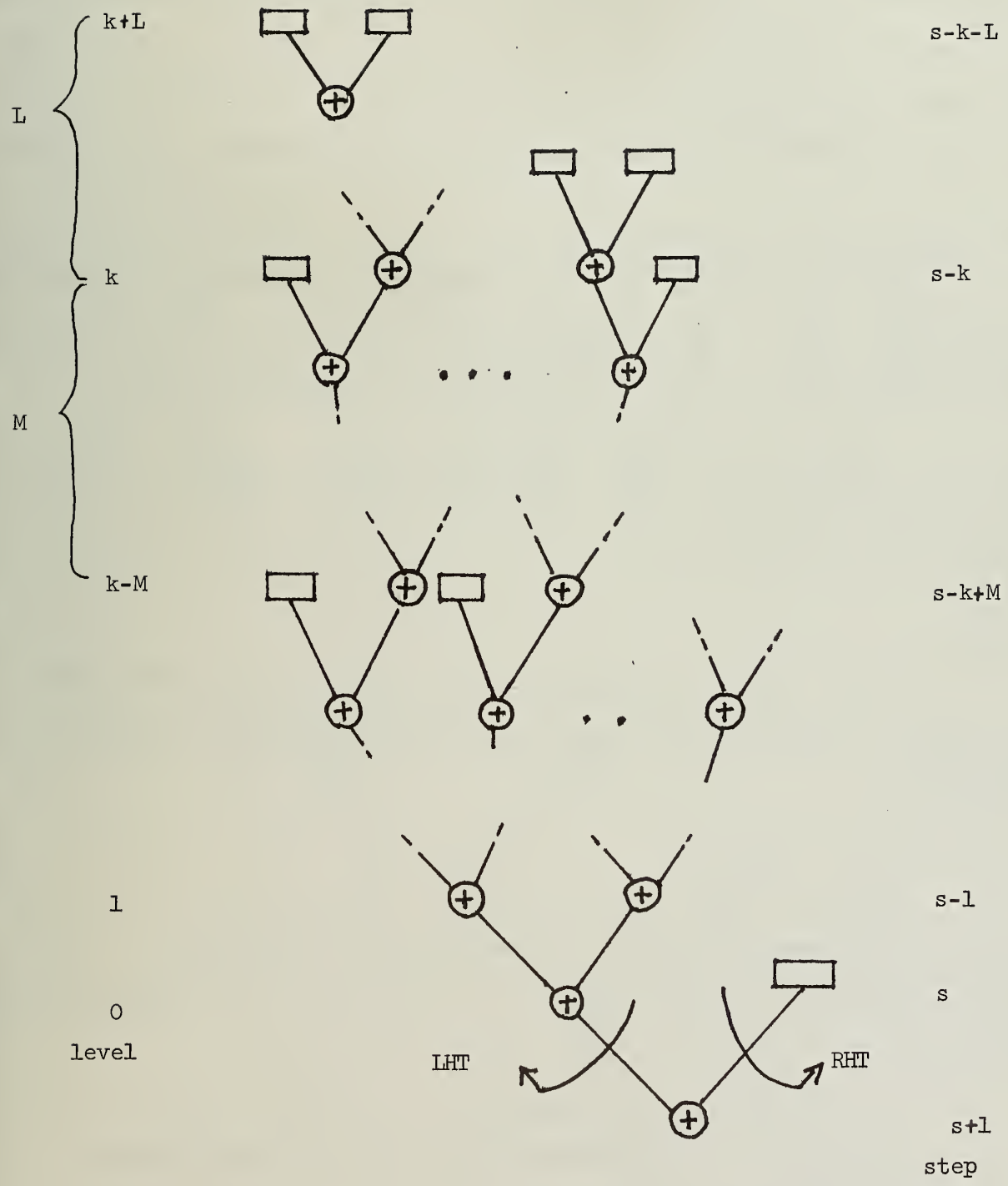


Fig 1. a computation tree, ▭ denotes a segment
and ⊕ denotes a binary addition operator

minimum degree of an x-term of segments in A_0 is greater than the maximum degree of an x-term of segments in A_1 for $2^k < q - 1 < 2^{k+1}$, and if $q - 1 = 2^{k+1}$ then a LOF computation tree is just a completely balanced tree (see Figure 2). It is important to notice that for given s and q, the LOF computation tree is unique.

Lemma 1

Let us assume $2^k < q - 1 \leq 2^{k+1}$ and LHT be arranged in any non-balanced computation tree. If a pair of segments is ℓ levels moved up then it increases the maximum degree which can be evaluated in (s + 1) steps by at least

$$(\alpha^{\ell} - 1)(2 - \alpha)N(s-k-1-j) + 1 > 0, \qquad 0 \leq j \leq L,$$

where α is an amplification factor defined by the ratio of $N(s-k-j) + 1$ and $N(s-k-1-j) + 1$, and $1 < \alpha(s) < 2$.

Proof:

Let us assume the computation tree of formula (3) is q-consistent and a pair of segments in A_{M+j} are ℓ levels moved up, and let $\leftarrow$ denote an assignment,

$$A_{M+j} \leftarrow A_{M+j} - 2, \qquad A_{M+j-1} \leftarrow A_{M+j-1} + 1,$$
$$A_{M+j-\ell} \leftarrow A_{M+j-\ell} + 2 \text{ and } \quad A_{M+j-\ell-1} \leftarrow A_{M+j-\ell-1} - 1.$$

Thus the improvement achieved by this rearrangement is given by

$$2(N(s-k-1-j+\ell) + 1) + (N(s-k-1-j+1) + 1) - 2(N(s-k-1-j) + 1) - (N(s-k-1-j+ +1)+1)$$
$$\geq (2\alpha^{\ell} + \alpha - \alpha^{\ell+1} - 2)(N(s-k-1-j) + 1)$$
$$= (\alpha^{\ell} - 1)(2 - \alpha)(N(s-k-1-j) + 1) > 0$$

where $\alpha = \alpha(s-k-1-j) = (N(s-k-j) + 1) / (N(s-k-j-1) + 1)$ and in general $\alpha(s+1) \geq \alpha(s)$ for relatively large s.

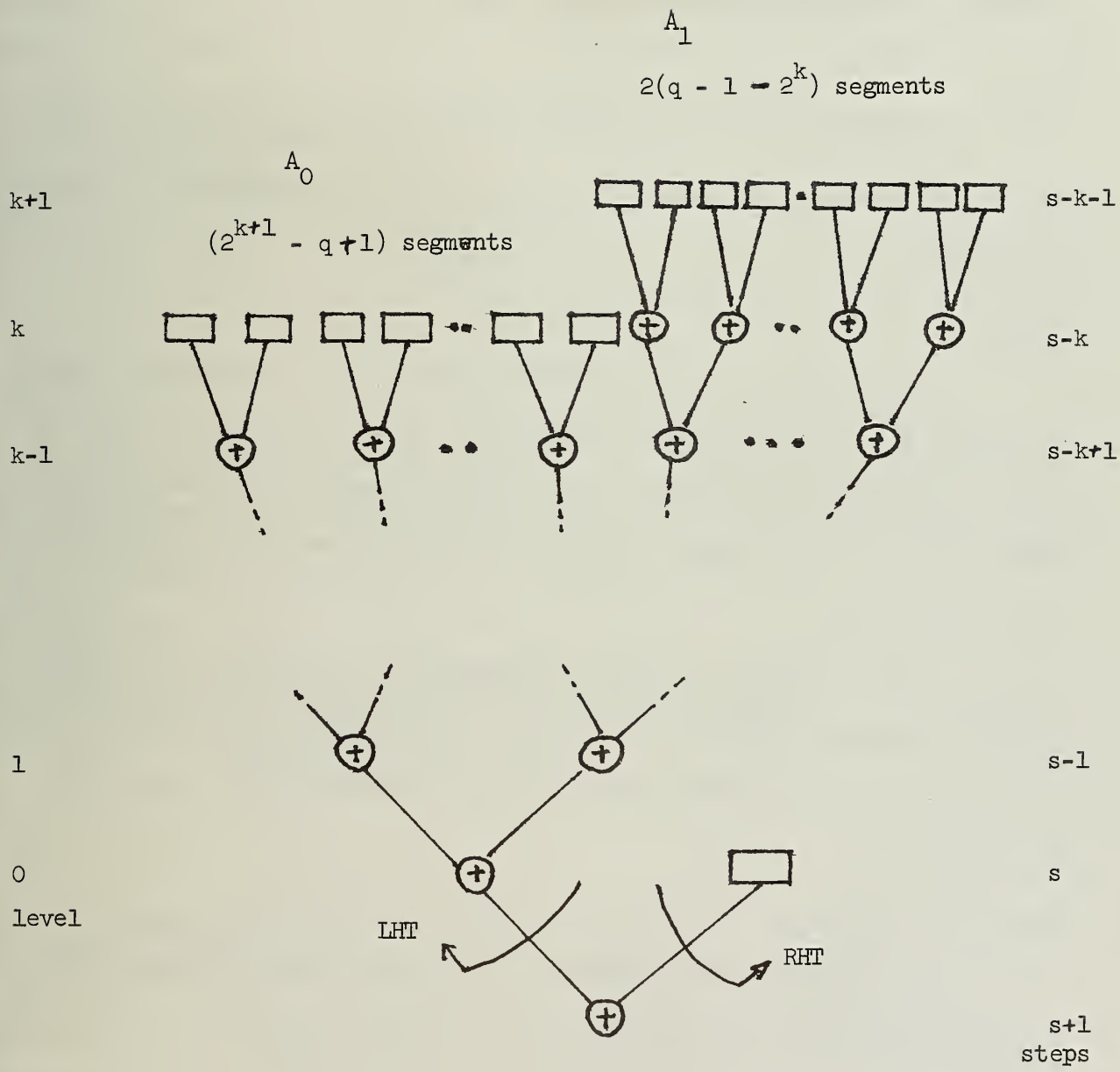


Fig 2. LOF computation tree

Lemma 1 implies that the maximum degree of polynomial evaluated in $(s + 1)$ steps by a balanced tree is greater than that achieved by a computation tree which is non-balanced with respect to q.

Lemma 2

For given q, $2^k < q-1 \leq 2^{k+1}$, if a balanced computation tree is q-consistent then the LOF computation tree is also q-consistent (the converse is not true).

Proof:

Let us assume the LOF computation tree is not q-consistent. Then the x-term, x^{m_j}, of the leftmost segment in A_1 is not consistent. Since m_j is not greater than the power m_i of the leftmost segment A_1 of any balanced computation tree, then such a balanced computation tree is not q-consistent.

(Q.E.D.)

Lemma 2 implies that the LOF computation tree can be q-consistent while the balanced computation tree is not q-consistent. In other words, the LOF computation tree is better than the regular balanced tree to solve our dual problem.

Lemma 3

The q-consistency condition, for $q \geq 3$ such that $2^k < q-1 \leq 2^{k+1}$, of an LOF computation tree at step $(s + 1)$ is given by

$$s - k - 2 \geq \lceil \log_2(N(s) + (2(q - 1 - 2^k) - 1)(N(s-k-2) + 1) + 1) \rceil$$

for $s \geq 7$. It is easy to see that 2-consistency always holds.

Proof:

See appendix A.

Lemma 4

Let us assume that the LHT of a computation tree at step (s + 1) is arranged as a non-LOF computation tree (can be balanced), and let the maximum degree of polynomial evaluated by such a computation tree be denoted by $\tilde{N}(s+1,\tilde{q})$. Then there exist at least one LOF computation tree of q-consistency, $q > \tilde{q}$ and $N(s+1,q) > \tilde{N}(s+1,\tilde{q})$.

Proof:

It is clear by the definition of q-consistency and the proof of Lemma 3.

(Q.E.D.)

Theorem 1

For $q \geq 3$ such that $2^k < q-1 \leq 2^{k+1}$, $k > 0$, if the q-consistency condition for the LOF computation tree holds at step (s + 1), then the maximum degree of polynomial evaluated in (s + 1) steps is given by

$$N(s+1,q) = N(s) + 2(q - 1 - 2^k)(N(s-k-2) + 1) + (2^{k+1} - q + 1)(N(s-k-1) + 1).$$

For $q = 2$, i.e., the folding method, gives the maximum degree.

$$N(s+1,2) = N(s) + N(s-1) + 1.$$

Proof:

Lemmas 1 through 4 lead us to the result that the maximum degree of polynomial evaluated at each step, $s > 0$, which is achieved by the LOF computation tree is greater than any other computation trees in LHT. By the definition of the LOF computation tree and the proof of Lemma 3, $q - 1 = A_0 + A_1$, $2^k < q - 1 \leq 2^{k+1}$ and $2^k = A_0 + A_1/2$, thus we get $A_0 = 2^{k+1} - q + 1$ and $A_1 = 2(q - 1 - 2^k)$.

(Q.E.D.)

Corollary 1

If a LOF computation tree is q-consistent then $N(s+1,q) > N(s+1,q-i)$ for $1 \leq i \leq q - 3$.

Proof:

It is easy to see that a q-consistent LOF computation tree is $(q - i)$-consistent for $1 \leq i \leq q - 2$. It is enough to show that $N(s+1,q) > N(s+1,q-1)$.

Case 1. (for $2^k = q - 2 < 2^{k+1}$)

$N(s+1,q) = N(s) + 2(q - 1 - 2^k)(N(s-k-2) + 1) + (2^{k+1} - q + 1)(N(s-k-1) + 1)$,

$N(s+1,q-1) = N(s) + 2^k(N(s-k-1) + 1)$. Thus

$$N(s+1,q) - N(s+1,q-1) = 2(N(s-k-2) + 1) - N(s-k-1) - 1$$
$$= (2 - \alpha)(N(s-k-2) + 1) > 0.$$

Case 2. (for $2^k < q - 2 < q - 1 \leq 2^{k+1}$)

$N(s+1,q) = N(s) + 2(q - 1 - 2^k)(N(s-k-2) + 1) + (2^{k+1} - q + 1)(N(s-k-1) + 1)$,

$N(s+1,q-1) = N(s) + 2(q - 2 - 2^k)(N(s-k-2) + 1) + (2^{k+1} - q + 2)(N(s-k-1) + 1)$.

Thus $$N(s+1,q) - N(s+1,q-1) = 2(N(s-k-2) + 1) - N(s-k-1) - 1$$
$$= (2 - \alpha)(N(s-k-2) + 1) > 0.$$

(Q.E.D)

3. ALGORITHMS

Corollary 1 implies that to maximize $N(s+1)$ we should choose the maximum q such that the LOF computation tree is q-consistent. From this, we get the following algorithm to generate $N(s+1)$ for $s > 2$.

Algorithm 1 (multi-folding method).

Step 1: Set $N(0) \leftarrow N(1) \leftarrow 0$, $N(2) \leftarrow 1$, $q \leftarrow 2$ and $s \leftarrow 2$.

Step 2: If the LOF computation tree is 3-consistent then go to Step 3. Set $q^* \leftarrow q$, $N(s+1,q^*) \leftarrow N(s) + N(s-1) + 1$, $s \leftarrow s + 1$ and if $s > M$, where M is an integer used to limit the number of steps, then stop, otherwise go back to Step 2.

Step 3: Let q^* be the maximum q such that the LOF computation tree is q-consistent, where $2^k < q - 1 \leq 2^{k+1}$.

Step 4: Set

$$N(s+1,q^*) \leftarrow N(s) + 2(q^* - 1 - 2^k)(N(s-k-2) + 1) + (2^{k+1} - q^* + 1)(N(s-k-1) + 1)$$

and $s \leftarrow s + 1$. If $s > M$ then stop, otherwise go back to Step 3.

So far, $P_{N(s-k-1)}(x)$ and $P_{N(s-k-2)}(x)$, the maximum degrees of polynomials evaluated in (s-k-1) and (s-k-2) steps, respectively, were assumed for each subpolynomial of the segments in A_0 and A_1 on LHT, respectively.

The q-consistency condition for a LOF computation tree given in Lemma 3; $s - k - 2 \geq \lceil \log_2 u \rceil$, where $u = N(s) + (2q^* - 2^{k+1} - 3)(N(s-k-2) + 1) + 1$, so we have information to increase $N(s+1)$. Let $\beta = 2^{s-k-2} - u \geq 0$ then we may expect γ improvement on $N(s+1,q^*)$ given by Algorithm 1 by forcing β to be zero and achieving $(q^* + 1)$-consistency. Such γ improvement is defined as

$$\gamma = \lambda(N(s-k-2) + \beta - N(s-k-1)) \quad \text{for } 2^k \leq q^* - 1 < 2^{k+1},$$

where λ is a function such that $\lambda(y) = 0$ for $y \leq 0$ and $\lambda(y) = y$ for $y > 0$. This leads to Algorithm 2.

Algorithm 2 (modified multi-folding method).

Step 1: Set $N(0) \leftarrow N(1) \leftarrow 0$, $N(2) \leftarrow 1$, $q \leftarrow 2$ and $s \leftarrow 2$.

Step 2: If the LOF computation tree is 3-consistent then go to Step 3. Set $q^* \leftarrow q$, $N(s+1, q^*) \leftarrow N(s) + N(s-1) + 1$, $s \leftarrow s + 1$ and if $s > M$ then stop, otherwise go back to Step 2.

Step 3: Let q^* be the maximum q such that the LOF computation tree is q-consistent, where $2^k < q - 1 \leq 2^{k+1}$.

Step 4: If $q^* - 1 = 2^{k+1}$ then go to Step 5. Set $\beta \leftarrow 2^{s-k-3} - N(s) - 1$, $\gamma \leftarrow N(s-k-3) + \beta - N(s-k-2)$ and if $\gamma > 0$ then set $\lambda \leftarrow 1$, otherwise $\lambda \leftarrow 0$. Go to Step 6.

Step 5: Set $\beta \leftarrow 2^{s-k-2} - N(s) - 2(q^* - 1 - 2^k)(N(s-k-2) + 1) - 1$ and $\gamma \leftarrow N(s-k-2) + \beta - N(s-k-1)$. If $\gamma \leftarrow 0$ then set $\lambda \leftarrow 1$, otherwise $\lambda \leftarrow 0$.

Step 6: Set $N(s+1, q^* + \lambda) \leftarrow N(s) + 2(q^* - 1 - 2^k)(N(s-k-2) + 1) + (2^{k+1} - q^* + 1)(N(s-k-1) + 1) + \lambda\gamma$ and $s \leftarrow s + 1$. If $s > M$ then stop, otherwise go back to Step 3.

4. COMPUTATION RESULTS AND THE BOUNDS

Table 1 shows the computation results of the maximum degree of polynomials evaluated by Muraoka's folding method, Algorithm 1, Algorithm 2 and the theoretical lower bound of the minimum required steps to evaluate polynomials of degree given by Algorithm 2. The amplification factor $\alpha(s)$, defined by the ratio between N(s) and N(s-1), approaches 2 as s increases. Our results agree with Brent's result that polynomials of degree $2^{k(k-1)/2}$ are evaluated in $k(k+1)/2$ steps, e.g., for $k = 5$ the degree is 1024 and s is 15 steps. Table 1 shows a polynomial of degree 1024 can be evaluated in 15 steps by our LOF computation tree methods (multi-folding and modified multi-folding) because by Algorithm 1 we can evaluate a polynomial of degree 1728 in 15 steps.

The following results from our algorithms may be more interesting than those of Brent. Remember that N(s) denotes the maximum degree of polynomial evaluated in s steps by the LOF computation tree (multi-folding or modified multi-folding methods). The details of derivations of the following results are in Appendix A. Any $s \geq 2$ can be denoted by either $k(k+1)/2 - 1$ for $k \geq 2$ or $k(k+1)/2$ for $k \geq 2$ or $k(k+1)/2 + i$ for $k \geq i + 1$, $i \geq 1$.

(1) For $k \geq 2$ and k is an integer

$$2^{k(k-1)/2} \geq N(k(k+1)/2 - 1) > 2^{k(k-1)/2} - 1,$$

(2) for $k \geq 2$

$$2^{k(k-1)/2 + 1} \geq N(k(k+1)/2) > 2^{k(k-1)/2},$$

(3) and for $k \geq i + 1$, $i \geq 1$

$$2^{k(k-1)/2 + i + 1} \geq N(k(k+1)/2 + i) > 2^{k(k-1)/2 + i}.$$

The above three lead to:

(1). for $s = k(k+1)/2 - 1$, $k \geq 2$,

$$2^{s} + 3/2 - (2(s+1))^{1/2} > N(s) > 2^{s} + 1/2 - (2(s+1))^{1/2},$$

(2). for $s = k(k+1)/2$, $k \geq 2$,

$$2^{s} + 3/2 - (2s)^{1/2} > N(s) > 2^{s} + 1/2 - (2s)^{1/2} \quad \text{and}$$

(3). for $s = k(k+1)/2 + i$, $i \geq 1$, $k \geq i + 1$,

$$2^{s} + 3/2 - (2(s-i))^{1/2} > N(s) > 2^{s} + 1/2 - (2(s-i))^{1/2}.$$

And we get the following.

Theorem 2 (dual property)

For any given $\delta > 0$, $N^{*}(s) \geq N(s) > 2^{s(1-\delta)}$, where $N^{*}(s)$ denotes the maximum degree of polynomial which can be evaluated in s steps, (i.e., theoretical maximum degree), and $\delta \approx (2/s)^{1/2}$ for all sufficiently large s.

Proof:

It is easily derived from dual properties.

(Q.E.D.)

We are now ready to talk about the primal problem, i.e., the minimum number of steps required to evaluate a polynomial of degree n.

(1). For $\log_2 n = k(k-1)/2 - 1$, $k \geq 2$,

$$T(P_n) \leq \log_2 n(1 + (2/(\log_2 n + 1))^{1/2}) + 1/2,$$

(2). for $\log_2 n = k(k-1)/2$, $k \geq 2$,

$$T(P_n) \leq \log_2 n(1 + (2/\log_2 n)^{1/2}) + 1/2 \quad \text{and}$$

(3). for $\log_2 n = k(k-1)/2 + i$, $k \geq i + 1$, $i \geq 1$,

$$T(P_n) \leq \log_2 n(1 + (2/(\log_2 n + i))^{1/2}) + 1/2.$$

Thus we get Theorem 3 which is identical to Brent's Theorem 2.

Theorem 3 (primal property)

For any given $\epsilon > 0$, $T(P_n) \leq (1 + \epsilon)\log_2 n$, for all sufficiently large n, where $T: P_n \rightarrow s$ and $\epsilon \approx (2\log_2 n)^{1/2}$.

Proof:

It is derived from primal properties.

(Q.E.D.)

Theorem 4

The number of operations required to evaluate a polynomial of degree n , where n is large, in less than or equal to $(1 + \epsilon)\log_2 n$ steps is given by $\#(P_n) < Cn$, where $C \approx 2^{\epsilon \log_2 n} > 2$.

Proof:

From Theorem 3, $\#(P_n) < 2^s - 1 \approx 2^{(1 + \epsilon)\log_2 n} = n2^{\epsilon \log_2 n}$.

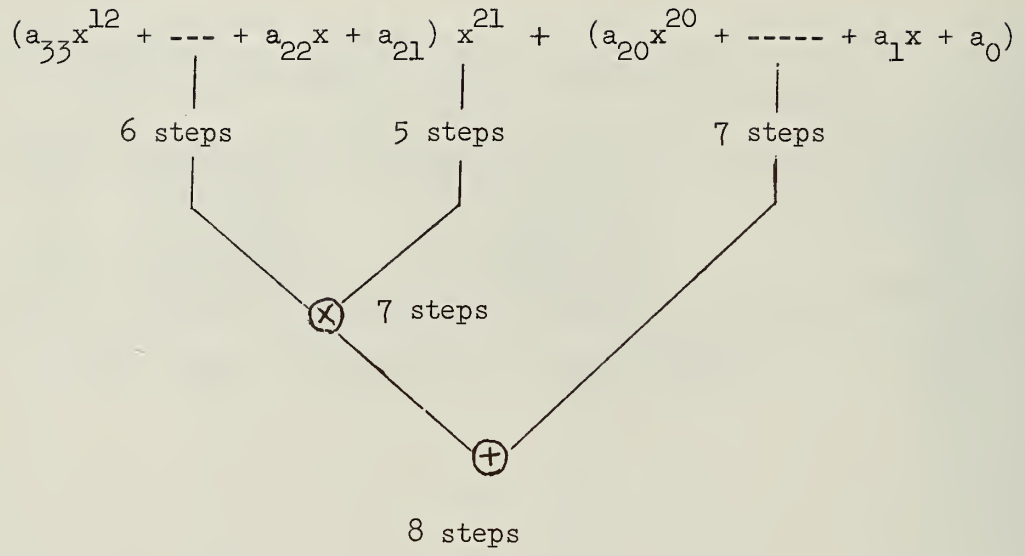
(Q.E.D.)

s	N(s) by folding method, q = 2	N(s) by Algorithm 1	q^*	N(s) by Algorithm 2	$q^* + \lambda$	lower bound
1	0	0	-	0	-	
2	1	1	2	1	2	2
3	2	2	2	2	2	3
4	4	4	2	4	2	4
5	7	7	2	7	2	4
6	12	12	2	12	2	5
7	20	20	2	20	2	6
8	33	36	3	36	3	7
9	54	62	3	62	3	7
10	88	104	3	104	3	8
11	143	183	4	183	4	9
12	232	320	4	320	4	10
13	376	572	5	572	5	11
14	609	992	5	992	5	11
15	986	1728	5	1728	5	12
16	1596	3059	6	3059	6	13
17	2583	5489	7	5489	7	14
18	4180	9767	7	9767	7	15
19	6764	17454	8	17454	8	16
20	10945	31286	9	31286	9	16
21	17710	55766	9	55915	10	17
22	28656	100946	11	101095	11	18
23	46367	182726	12	182875	12	19
24	75024	330690	13	330839	13	20
25	121392	602724	15	602873	15	21
26	196417	1096509	16	1096807	16	22
27	317810	1988781	17	1991463	17	22
28	514228	3614520	18	3619735	18	23
29	832039	6595653	20	6603699	20	24
30	1346268	12060524	22	12071699	22	25
31	2178308	22114723	24	22129325	24	26
32	3524577	40639343	26	40738153	27	27
33	5702886	74910711	29	75027401	29	28
34	9227464	138188692	32	138391057	32	29
35	14930331	253853364	33	254222609	33	29

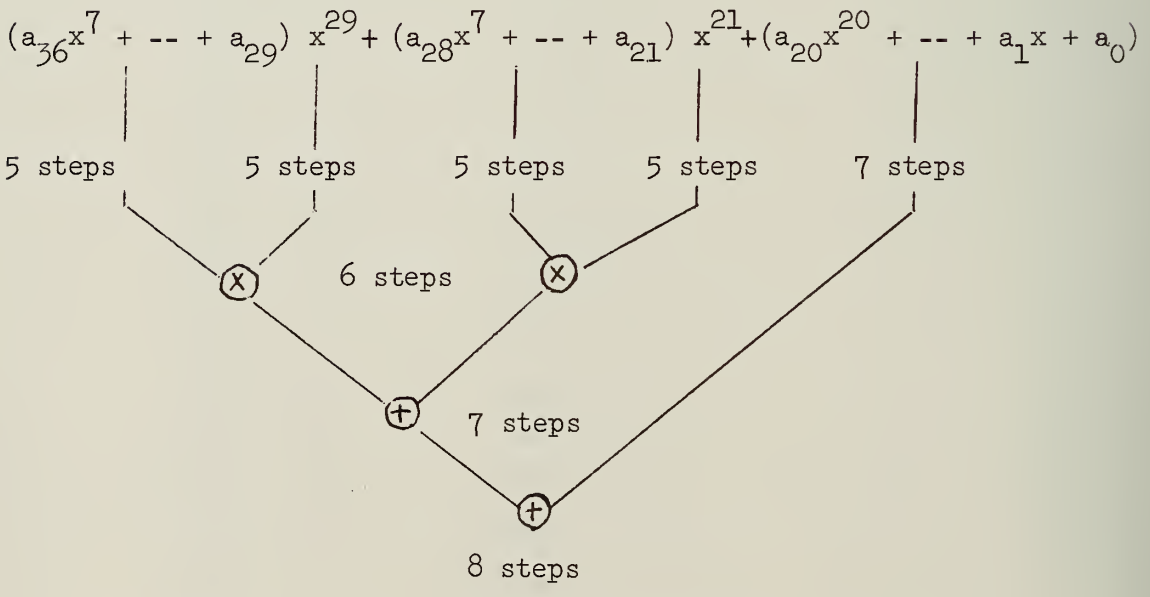
Table 1. Maximum degrees of polynomials at each step .

s	N(s) by folding method, $q = 2$	N(s) by Algorithm 1	q^*	N(s) by Algorithm 2	$q^* + \lambda$	lower bound
36	24157816	466181068	35	466812558	35	30
37	39088168	857771783	38	858785453	38	31
38	63245985	1583499885	42	1585050551	42	32

Table 1. Maximum degrees of polynomials st each step .

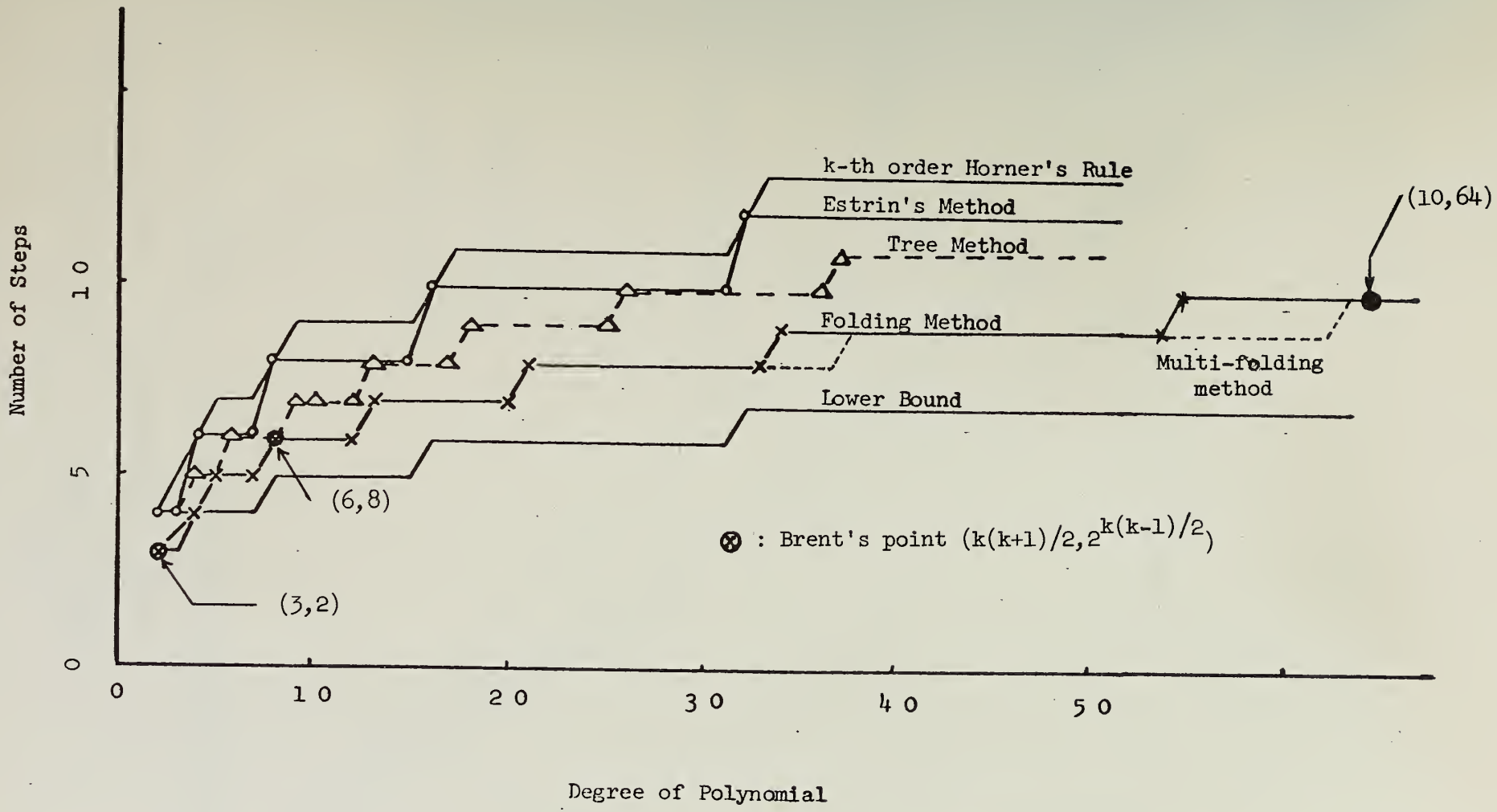


(a). The computation tree for Muraoka's folding method at step 8 (which evaluates a polynomial of degree 33).

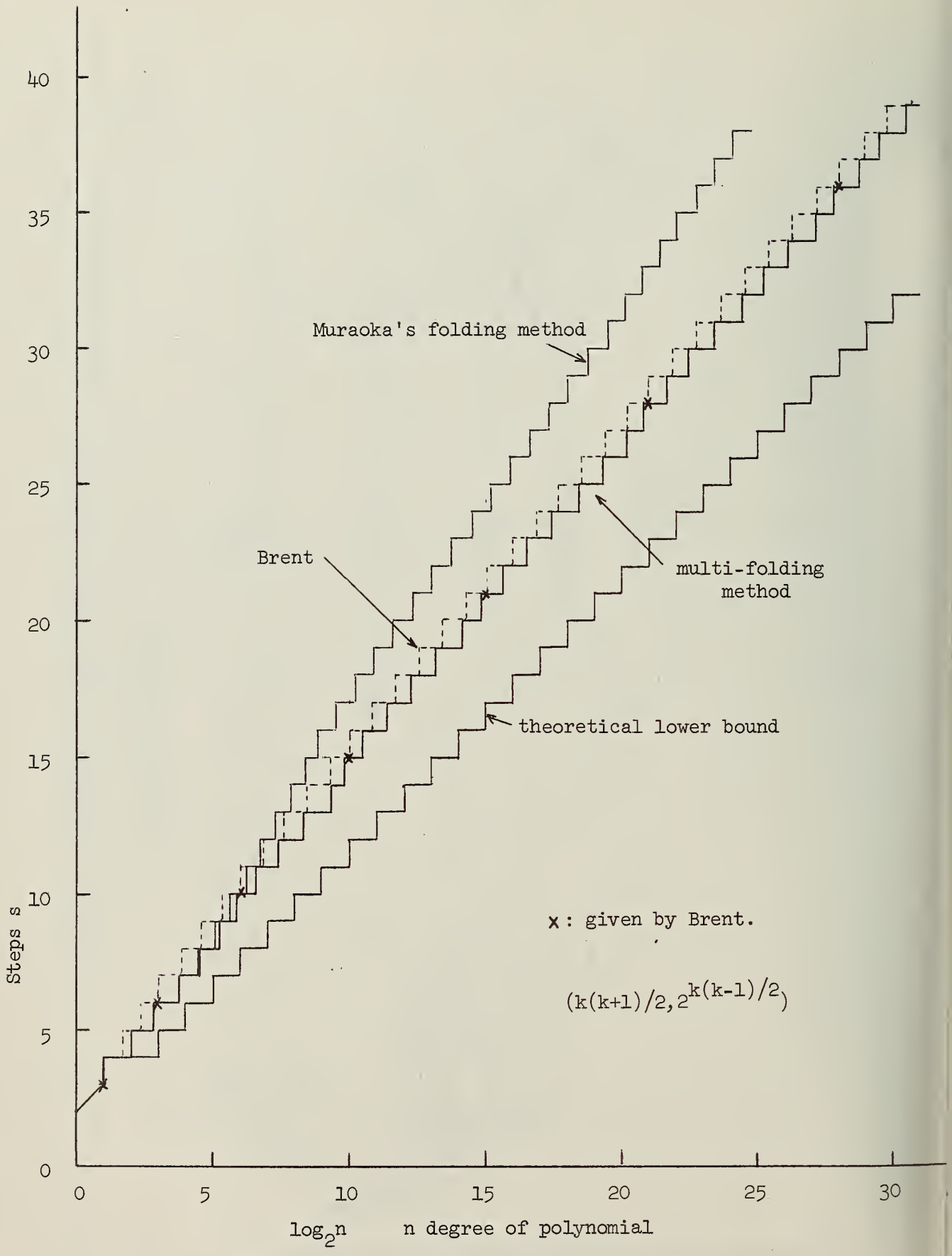


(b). The computation tree for multi-folding method at step 8 (which evaluates a polynomial of degree 36).

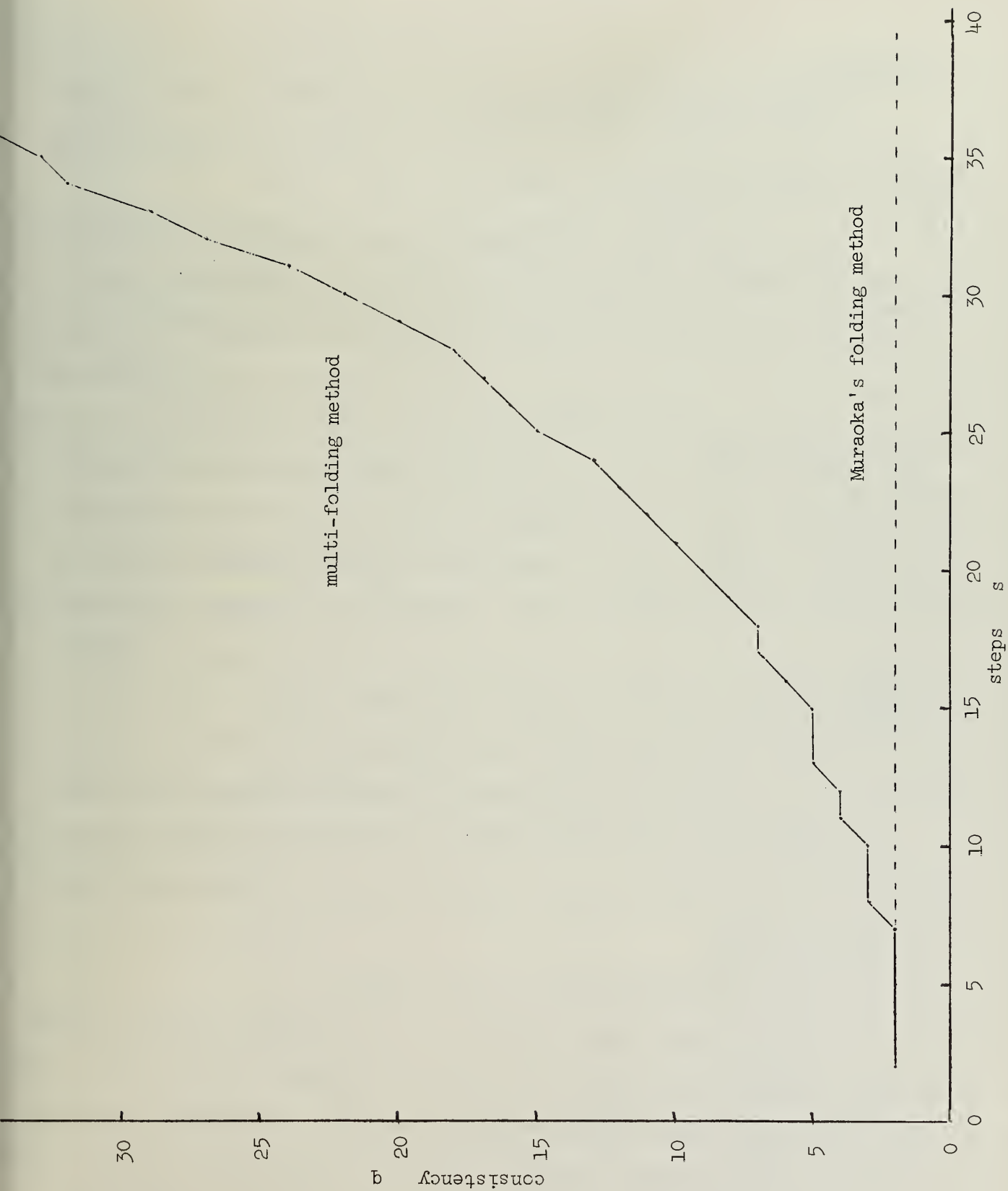
Fig. 3 Examples of computation trees at steps 8.



Graph 1. Comparison of five parallel polynomial computation methods.



Graph 2. Steps, s v.s. Polynomial degree n



Graph 3. Consistency q v.s. steps s.

5. CONCLUSION

Graph 2 shows the maximum degree of polynomial which can be evaluated in a given number of steps by the LOF computation tree. Brent's result is also plotted.

Let us compare Brent's result(his Theorem 1) which says that a polynomial of degree $2^{k(k-1)/2}$ can be evaluated in less than or equal to $k(k+1)/2$ steps, $k \geq 2$, with our result that a polynomial of degree greater than $2^{k(k-1)/2}$ can be evaluated in $k(k+1)/2$ steps. These two results are slightly different since the former statement does not say that there exists at least one polynomial of degree higher than $2^{k(k-1)/2}$ whose evaluation can be accomplished in $k(k+1)/2$ steps. This statement is also clear from Graphs 1 and 2. Our dual problem approach becomes more reasonable than the primal problem approach because a set of polynomials can be evaluated in s steps and such a set is ordered with respect to polynomial degree. Moreover it is easy to see that the dual property implies the primal property, while the converse is not true.

Some of the consequences of this paper are:

(1). We found the maximum possiblly evaluable degree of polynomial for each number of steps, i.e., the dual property for not only a particular s but for all $s \geq 1$, while Brent's primal property is applied only for descrete degrees or at descrete steps.

(2). Dual property implies primal property.

(3). We found simple computation trees called LOF, or multi-folding, to achieve Brent's upper bound and to approach the theoretical lower bound. This means that we have found a simple scheduling algorithm to evaluate a polynomial of degree n whose evaluation requires a number of steps close to the lower bound.

(4). We found ϵ as a function of $(2/\log_2 n)^{1/2}$ for the upper bound

$(1 + \epsilon)\log_2 n$ of the primal problem.

(5). We found δ as a function of $(2/s)^{1/2}$ for the lower bound $2^{s(1-\delta)}$ of the dual problem.

(6). The upper bound of the number of operations required to evaluate a polynomial of degree n in $(1 + \epsilon)\log_2 n$ steps is given as Cn, $C > 2$.

Furthermore, by considering Brent's statement "Theorem 1 is used to obtain an upper bound for t(n), even if n is not a power of 2", we get the following.

$$N_B(s) \geq 2^{s + 1/2 - (2s)^{1/2}}, \text{ for integer } s \geq 2.$$

Notice that the equality should be included, i.e., $\geq$, while our dual results do not, i.e., $>$ (see dual properties developped ealier). The maximum degree of polynomials derived by the above dualization of Brent's Theorem 2, $N_B(s)$, for each step s is plotted in the Graph 2. By the comparison of our results and his, we have

(7). the upper bound of the steps required to evaluate polynomials by our multi-folding method is lower than or equal to the Brent's, and the lower bound of the degree of polynomials evaluated at each step s, i.e., the maximum evaluable degree of polynomials, is higher than Brent's by a factor of μ, where $1.5 < \mu < 2$.

LIST OF REFERENCES

1. Brent, R., "On the addition of binary numbers", IEEE Transaction on Computers, August 1970, pp. 758-759.

2. Dorn, W. S., "Generalizations of Horner's Rule for Polynomial Evaluation", IBM Journal of Research and Development, 6, April 1962, pp. 239-245.

3. Estrin, G., "Organization of Computer Systems -- the Fixed plus Variable Structure Computer", Proc. of Western Joint Computer Conference, May 1960, pp. 33-40.

4. Muraoka, Y., "Parallelism Exposure and Exploitation in Programs", Ph.D. Thesis, University of Illinois, 1971, pp. 33-41.

5. Pan, V. Ya., "Methods of Computing Values of Polynomials", Russian Mathematical Surveys, 21, January-February 1966, pp. 105-136.

6. Winograd, S., "On the Time Required to Perform Addition", JACM, 12, April 1965, pp. 277-285.

APPENDIX A

PROOFS OF MAIN PROPERTIES

Proof of Lemma 3.

If $s - 1 \geq \lceil \log_2(N(s) + 1) \rceil$ then $N(s+1,2) = N(s) + N(s-1) + 1$.

If $s - 2 \geq \lceil \log_2(N(s) + N(s-2) + 2) \rceil$ then

$$N(s+1,3) = N(s) + 2(N(s-2) + 1).$$

If $s - 2 \geq \lceil \log_2(N(s) + 2(N(s-3) + 1) + 1) \rceil$ and if

$s - 3 \geq \lceil \log_2(N(s) + (N(s-3) + 1) + 1) \rceil$ then

$$N(s+1,4) = N(s) + 2(N(s-3) + 1) + (N(s-2) + 1).$$

If $s - 3 \geq \lceil \log_2(N(s) + 3(N(s-3) + 1) + 1) \rceil$ then

$$N(s+1,5) = N(s) + 4(N(s-3) + 1).$$

If $s - 3 \geq \lceil \log_2(N(s) + 2(N(s-4) + 1) + 2(N(s-3) + 1) + 1) \rceil$ and if

$s - 4 \geq \lceil \log_2(N(s) + (N(s-4) + 1) + 1) \rceil$ then

$$N(s+1,6) = N(s) + 2(N(s-4) + 1) + 3(N(s-3) + 1).$$

It is important to see that "If condition" of the very first case always hold. In general we get the following properties.

Property 1.

If $q - 1 = 2^k$ and if $s - k - 1 \geq \lceil \log_2(N(s) + (2^k - 1)(N(s-k-1) + 1) + 1) \rceil$ then $N(s+1,q) = N(s) + 2^k(N(s-k-1) + 1)$.

Property 2.

If $2^k < q - 1 < 2^{k+1}$,

$s - k - 1 \geq \lceil \log_2(N(s) + 2(q - 1 - 2^k)(N(s-k-2) + 1) + (2^{k+1} - q)(N(s-k-1) + 1) + 1) \rceil$ and if $s - k - 2 \geq \lceil \log_2(N(s) + (2(q - 1 - 2^k) - 1)(N(s-k-2) + 1) + 1) \rceil$ then

$N(s+1,q) = N(s) + 2(q - 1 - 2^k)(N(s-k-2) + 1) + (2^{k+1} - q + 1)(N(s-k-1) + 1)$.

Property 3.

If the second condition of Property 2 holds then the first condition always holds.

Proof:

$$N(s) + 2(q - 1 - 2^k)(N(s-k-2) + 1) + (2^{k+1} - q)(N(s-k-1) + 1) + 1$$

$$< 2N(s) < 2(N(s) + (2(q - 1 - 2^k) - 1)(N(s-k-2) + 1) + 1).$$

Thus from the above arguments, the q-consistency condition ($q \geq 3$) for the LOF computation tree is given by

$$s - k - 2 \geq \lceil \log_2(N(s) + (2(q - 1 - 2^k) - 1)\ (N(s-k-2) + 1) + 1) \rceil$$

where $2^k \leq q - 1 \leq 2^{k+1}$.

(Q.E.D.)

Dual Properties.

(1) At steps $s = (k(k + 1))/2 - 1$, for $k \geq 2$,

$$\log_2 N(s) + 2 > \lceil \log_2 N(s) \rceil + 1 = (k(k - 1))/2 + 1 \geq \log_2 N(s) + 1.$$

By rearrangement, we get

$$(k(k - 1))/2 \geq \log_2 N(s) > (k(k - 1))/2 - 1$$

and this yields to:

$$2^{(k(k - 1))/2} \geq N(s) > 2^{(k(k - 1))/2 - 1}.$$

(2) At steps $s = (k(k + 1))/2$, for $k \geq 2$ (i.e., at Brent's points),

$$\log_2 N(s) + 2 > \lceil \log_2 N(s) \rceil + 1 = (k(k - 1))/2 + 2 \geq \log_2 N(s) + 1.$$

By rearrangement, we get

$$(k(k - 1))/2 + 1 \geq \log_2 N(s) > (k(k - 1))/2$$

and this yields to:

$$2^{(k(k - 1))/2 + 1} > N(s) > 2^{(k(k - 1))/2}.$$

(3) At steps $s = (k(k + 1))/2 + i$, for $k \geq i + 1$, $i \geq 1$,

$$\log_2 N(s) + 2 > \lceil \log_2 N(s) \rceil + 1 = (k(k - 1))/2 + i + 2 \geq \log_2 N(s) + 1$$

By rearrangement, we get

$$(k(k - 1))/2 + i + 1 \geq \log_2 N(s) > (k(k - 1))/2 + i$$

and this yields:

$$2^{(k(k - 1))/2 + i + 1} \geq N(s) > 2^{(k(k - 1))/2 + i}.$$

Primal Properties.

These properties are derived directly from those dual properties discussed above.

APPENDIX B

SOME OTHER PROPERTIES

The following two properties are important and are given without proofs.

Lemma 5

Let $P_{n_1}(x)$ and $P_{n_2}(x)$ be two polynomials, where $n_1 > n_2$. If s_1 and s_2 are the minimum steps required to evaluate P_{n_1} and P_{n_2}, respectively, then $s_1 \geq s_2$ (the converse is not true).

Lemma 6

Let s_1 and s_2 be the minimum steps required to evaluate polynomials $P_{n_1}(x)$ and $P_{n_2}(x)$, respectively. If $s_1 > s_2$ then $n_1 > n_2$ (the converse is not true).

Theorem 5

Muraoka's folding method and the minimum steps required to evaluate polynomials achieved by exhaustive division of $P_n(x)$ into 2 segments, called 2-cut, are equivalent, i.e., both methods give the same maximum degree of polynomials that can be evaluated in a given number of steps.

Proof:

Let s_f and s_b denote the minimum number of steps required to evaluate a polynomial of degree n by the folding method and exhaustive 2-cut method, respectively, and assume T is the function T: $P_n(x) \rightarrow s$. If $s_f \leq s_b$ we have

$$Q_{n-\gamma}(x)x + P_{\gamma-1}(x) \in \left\{ Q_{n-i}(x)x^i + P_{i-1}(x) \mid 1 \leq i \leq n \right\},$$

therefore, $s_f \geq s_b = \min \left\{ T(Q_{n-i}(x)x^i + P_{i-1}(x)) \mid 1 \leq i \leq n \right\}$, where γ is given by the folding method.

Now consider the case $s_f \geq s_b$. Let us assume $s_f < s_b$, then there exists k such that $s_f > T(Q_{n-k}(x)x^k + P_{k-1}(x))$. It is also obvious that $s_f - 2 \geq T(P_{k-1}(x))$ and $s_f - 3 \geq \log_2 k$ for $k < \gamma$ ($k \geq \gamma$ is not valid from Lemma 4). But for such k $s_f - 3 \geq T(Q_{n-k}(x))$ must hold, while $s_f - 2 \geq T(Q_{n-\gamma}(x))$, since $k < \gamma$ implies $\deg(Q_{n-k}(x)) \geq \deg(Q_{n-\gamma}(x))$ and by Lemma 4 we get $T(Q_{n-k}(x)) \geq T(Q_{n-\gamma}(x))$ which leads us to a contradiction.

(Q.E.D.)

Theorem 6

The minimum steps s_f required to evaluate a polynomial of degree n by Muraoka's folding method is upper bounded by $(1 + C_1)\log_2 n + C_2$ for relatively large n, where C_1 (≈ 0.44) and C_2 (≈ 0) are constants.

Proof:

Because the sequence $\{N(s) + 1 \mid s \geq 0\}$ forms a Fibonacci sequence, the amplification factor $\alpha(s)$ for relatively large s approaches $(1 + 5^{1/2})/2$ (≈ 1.62), and then we get the result.

(Q.E.D.)

For example the minimum steps s_f required to evaluate $P_n(x)$ by the folding method for $2 \leq n \leq 200$ are bounded by

$$\lceil (1 + 0.38)\log_2 n \rceil + 1 \leq s_f \leq \lceil (1 + 0.4)\log_2 n \rceil + 1.$$

Theorem 7

The minimum number of operations required to evaluate a polynomial of degree n, $N(s) < n \leq N(s) + N(s-1) + 1$, by Muraoka's folding method is given by the following recurrence formula;

$$\#(P_{N(s)+i+1}(x)) = \#(P_i(x)) + 3 + \#(P_{N(s)}(x))$$

for $i = 0, 1, \ldots, N(s-1)$, and for $s = 1, 2, \ldots$, where $\#: P_n(x) \rightarrow I$, and $\#(P_n) \geq C'n$ where $C' \approx 2.6$.

Proof:

It is obvious from the computation tree of Muraoka's folding method, or from our 2-consistent LOF computation tree.

(Q.E.D.)

In table 2, the computation result given by Theorem 7 is listed.

Lemma 7

Let us assume $P_{N(s)}(x)$ denotes the polynomial of the maximum degree whose evaluation requires at least s steps by "any" computation tree, then

$$T(P_{N(s)}(x)x^i) \geq s + 1 \text{ for } i \geq 1.$$

Proof:

It is enough to show that $T(xP_{N(s)}(x)) \geq s + 1$. Assume $xP_{N(s)}$ can be evaluated in s steps. If $xP_{N(s)}$ can be evaluated by our multi-folding method then $P_{N(s)}$ should be evaluated by $(s - 1)$ steps, which leads to contradiction. If the polynomial can be evaluated as

$$\ldots + \ldots + \ldots + a_1x^2 + a_0x$$

since the evaluation of a_1x^2 and a_0x require 2 and 1 steps, respectively, it is clear to see that we can evaluate $P_{N(s)}x + c$, c is constant, in s steps. Furthermore, $P_{N(s)}$ is, by assumption, the polynomial of the maximum degree whose evaluation requires s steps, thus we have a contradiction. Assume the polynomial is evaluated as

$$\cdots + \cdots\cdots + \cdots\cdots + (\cdots\cdots)x^3 + (a_1x + a_0)x$$

we need 3 steps to evaluate the last term, but we can also evaluate $a_1x^2 + a_0x + c$ in 3 steps, which means we can evaluate $P_{N(s)} + c$ in s steps and leads us to a contradiction in our assumption.

(Q.E.D.)

n	$\#(P_n)$	n	$\#(P_n)$	n	$\#(P_n)$	n	$\#(P_n)$
0	0	35	91	70	183	105	273
1	2	36	94	71	186	106	275
2	5	37	97	72	188	107	278
3	8	38	99	73	191	108	280
4	10	39	102	74	193	109	283
5	13	40	104	75	196	110	286
6	15	41	107	76	199	111	288
7	18	42	110	77	201	112	291
8	21	43	112	78	203	113	294
9	23	44	115	79	205	114	296
10	26	45	118	80	207	115	299
11	29	46	120	81	210	116	301
12	31	47	123	82	212	117	304
13	34	48	125	83	215	118	307
14	36	49	128	84	218	119	309
15	39	50	131	85	220	120	312
16	42	51	133	86	223	121	315
17	44	52	136	87	226	122	317
18	47	53	138	88	228	123	320
19	49	54	141	89	231	124	322
20	52	55	144	90	233	125	325
21	55	56	146	91	236	126	328
22	57	57	149	92	239	127	330
23	60	58	152	93	241	128	333
24	63	59	154	94	244	129	335
25	65	60	157	95	246	130	338
26	68	61	159	96	249	131	341
27	70	62	162	97	252	132	343
28	73	63	165	98	254	133	346
29	76	64	167	99	257	134	349
30	78	65	170	100	260	135	351
31	81	66	173	101	262	136	354
32	84	67	175	102	265	137	356
33	86	68	178	103	267	138	359
34	89	69	180	104	270	139	362

Table 2. The number of operations required to evaluate polynomial of degree n by the folding method.

APPENDIX C
PL/1 PROGRAMS

```
DEGREE: PROCEDURE OPTIONS(MAIN);
     DCL N(0:51) BIN FIXED(31),
         S BIN FIXED,
         M BIN FIXED(31),
        DELTA BIN FIXED(31),
         TOWK(0:35) BIN FIXED(31);
     N(0)=0;
     N(1)=0;
     N(2)=1;
     N(3)=2;
     N(4)=4;
     N(5)=7;
     N(6)=12;
     N(7)=20;
     TOWK(0)=1;
     DO I=1 TO 30;
        TOWK(I)=2*TOWK(I-1);
     END;
     NC=3;
     DO S=7 TO 50;
   L: DO I=0 TO 10;
        IF (TOWK(I)<NC-1)&(NC-1<=TOWK(I+1))
        THEN DO; J=I; GO TO OUT; END;
     END;
OUT:M=N(S)+(2*(NC-1-TOWK(J))-1)*(N(S-J-2)+1)+1;
     IF TOWK(S-J-2)>=M
                THEN DO: NCD=NC; NC=NC+1; K=J; GO TO L: END;
     N(S+1)=N(S)+2*(NCD-1-TOWK(K))*(N(S-K-2)+1)
               +(TOWK(K+1)-NCD+1)*(N(S-K-1)+1);
     PUT SKIP DATA(N(S+1),NCD);
     NC=NC-1;
     END;
END DEGREE;
```

```
PRINT:
    PUT SKIP DATA(N(S+1),NCD);
    NC=NC-1;
    END;
END DEGREE;
```

```
DEGREE: PROCEDURE OPTIONS(MAIN);
    DCL N(0:51) BIN FIXED(31),
        S BIN FIXED,
        M BIN FIXED(31),
       DELTA BIN FIXED(31),
        TOWK(0:35) BIN FIXED(31);
    N(0)=0;
    N(1)=0;
    N(2)=1;
    N(3)=2;
    N(4)=4;
    N(5)=7;
    N(6)=12;
    N(7)=20;
    TOWK(0)=1;
    DO I=1 TO 30;
       TOWK(I)=2*TOWK(I-1);
    END;
    NC=3;
    DO S=7 TO 50;
  L: DO I=0 TO 10;
       IF (TOWK(I)<NC-1)&(NC-1<=TOWK(I+1))
       THEN DO; J=I; GO TO OUT; END;
    END;
OUT:M=N(S)+(2*(NC-1-TOWK(J))-1)*(N(S-J-2)+1)+1;
    IF TOWK(S-J-2)>=M
                  THEN DO: NCD=NC: NC=NC+1; K=J; GO TO L; END;
    IF TOWK(K+1)=NCD-1 THEN GO TO OUT1;
    DELTA=TOWK(S-K-2)-N(S)-2*(NCD-1-TOWK(K))*(N(S-K-2)+1)-1;
    IF N(S-K-1)>=N(S-K-2)+DELTA
    THEN DO;
    N(S+1)=N(S)+2*(NCD-1-TOWK(K))*(N(S-K-2)+1)
               +(TOWK(K+1)-NCD+1)*(N(S-K-1)+1);
    GO TO PRINT;
       END;
    ELSE DO;
    N(S+1)=N(S)+(2*(NCD-1-TOWK(K))+1)*(N(S-K-2)+1)
              +(TOWK(K+1)-NCD)*(N(S-K-1)+1)+DELTA;
    NCD=NCD+1;
    GO TO PRINT;
       END;
OUT1:
    DELTA=TOWK(S-K-3)-N(S)-1;
    IF N(S-K-2)>=N(S-K-3)+DELTA
    THEN DO;
    N(S+1)=N(S)+TOWK(K+1)*(N(S-K- 2)+1);
    GO TO PRINT;
       END;
    ELSE DO;
    N(S+1)=N(S)+(TOWK(K+1)-1)*(N(S-K-2)+1)+N(S-K-3)+DELTA;
    NCD=NCD+1:
       END;
```

```
BISECT: PROCEDURE OPTIONS(MAIN);
```

```
BISECT: PROCEDURE OPTIONS(MAIN);
    DCL DPN BIN FIXED;
    GET LIST(DPN);
    BEGIN;
    DCL (A,B,C,S1,S2,MINISTEP) BIN FIXED,
        STEP(0:DPN) BIN FIXED;
    STEP(0)=0; STEP(1)=2;STEP(2)=3;
    DO N=3 TO DPN;
       MINISTEP=100;
       DO I=1 TO N;
          S1=STEP(I-1);
          A=CEIL(LOG2(I));
          B=STEP(N-I);
          S2=MAX(A,B)+1;
          C=MAX(S1,S2)+1;
          IF C<MINISTEP THEN DO; MINISTEP=C; MI=I; END;
       END;
       STEP(N)=MINISTEP;
         LB=CEIL(LOG2(2*N+1));
       PUT SKIP DATA(N,MINISTEP,MI,LB);
    END;
    END;
END BISECT;
```

```
TRI_SEC: PROCEDURE OPTIONS(MAIN);
```

```
TRI_SEC: PROCEDURE OPTIONS(MAIN);
    DCL NN BIN FIXED;
    GET LIST(NN);
    BEGIN;
    DCL (A,B,C,D,S1,S2,S3) BIN FIXED,
        STEP(0:NN) BIN FIXED;
    STEP=100;
    STEP(0)=0; STEP(1)=2;
    DO N=2 TO NN;
    K=N-1;
    DO  I=1 TO K;
         S1=STEP(I-1);
    DO J=I+1 TO N;
         A=STEP(J-I-1);
         B=CEIL(LOG2(I));
         S2=MAX(A,B)+1;
         A=STEP(N-J);
         B=CEIL(LOG2(J));
         S3=MAX(A,B)+1;
         A=MAX(S1,S2,S3);
         IF A=S1 THEN DO; C=MAX(S2,S3)+1; D=MAX(S1,C)+1;
        IF D<=STEP(N) THEN DO; STEP(N)=D; MI=I; MJ=J; END;
         GO TO OUT; END;
         IF A=S2 THEN DO; C=MAX(S1,S3)+1; D=MAX(S2,C)+1;
        IF D<=STEP(N) THEN DO; STEP(N)=D; MI=I; MJ=J; END;
         GO TO OUT; END;
         C=MAX(S1,S2)+1; D=MAX(S3,C)+1;
        IF D<=STEP(N) THEN DO; STEP(N)=D; MI=I; MJ=J; END;
    OUT:
    END;
    END;
    PUT SKIP DATA(STEP(N),MI,MJ);
    END;
    END;
END TRI_SEC;
```

```
EXHAUST: PROCEDURE OPTIONS(MAIN);
DCL NI(50) BIN FIXED, /* INDEX REGS */
    H(0:100) BIN FIXED,
    STEP(0:50) BIN FIXED,
    P(50) BIN FIXED, /* CUT POSITION */
    (N,NN,NC,A,B,SR,COUNT) BIN FIXED;

UP_DATE_H: PROCEDURE;
    H=0;
    SR=STEP(NI(1)-1);
    H(SR)=1;
    IF NC>=3 THEN
    DO I=2 TO NC_1;
       A=STEP(NI(I)-NI(I-1)-1);
       B=CEIL(LOG2(NI(I-1)));
       SR=MAX(A,B)+1;
       H(SR)=H(SR)+1;
    END;
    A=STEP(N-NI(NC_1));
    B=CEIL(LOG2(NI(NC_1)));
    SR=MAX(A,B)+1;
    H(SR)=H(SR)+1;
    COUNT=0; I=0;
  L:IF H(I)>0 THEN
    DO; IF H(I)>1 THEN DO; H(I+1)=H(I+1)+1; H(I)=H(I)-2; COUNT=COUNT+1;
                           IF COUNT<NC_1 THEN GO TO L;
                           M=I+1; GO TO FINE;
                       END;
        DO J=I+1 TO ND;
           IF H(J)>0 THEN DO; H(J+1)=H(J+1)+1; H(I)=H(I)-1;
                              H(J)=H(J)-1; COUNT=COUNT+1;
                              IF COUNT<NC_1 THEN GO TO L;
                              M=J+1; GO TO FINE;
                          END;
        END;
    END;
    I=I+1;
    GO TO L;
    FINE:
                    IF M<=STEP(N) THEN DO; STEP(N)=M; MC=NC;
                     DO K=1 TO NC_1; P(K)=NI(K); END; END;
    RETURN;
END UP_DATE_H;

    GET LIST(NN);
    STEP=100; NI=0;
    STEP(0)=0; STEP(1)=2; STEP(2)=3;
    DO N=3 TO NN;
    ND=N+N;
    P=0;
    IF N>27 THEN MNC=9; ELSE MNC=N/3+1;
LO :DO NC=2 TO MNC;
```

```
   NC_1=NC-1;
L1 :DO NI(1 )=1 TO N-NC+2;
    IF NC=2  THEN DO; CALL UP_DATE_H;
                      GO TO OUT1 ; END;
L2 :DO NI(2 )=NI(1 )+1 TO N;
    IF NC=3  THEN DO; CALL UP_DATE_H;
                      GO TO OUT2 ; END;
L3 :DO NI(3 )=NI(2 )+1 TO N;
    IF NC=4  THEN DO; CALL UP_DATE_H;
                      GO TO OUT3 ; END;
L4 :DO NI(4 )=NI(3 )+1 TO N;
    IF NC=5  THEN DO; CALL UP_DATE_H;
                      GO TO OUT4 ; END;
L5 :DO NI(5 )=NI(4 )+1 TO N;
    IF NC=6  THEN DO; CALL UP_DATE_H;
                      GO TO OUT5 ; END;
L6 :DO NI(6 )=NI(5 )+1 TO N;
    IF NC=7  THEN DO; CALL UP_DATE_H;
                      GO TO OUT6 ; END;
L7 :DO NI(7 )=NI(6 )+1 TO N;
    IF NC=8  THEN DO; CALL UP_DATE_H;
                      GO TO OUT7 ; END;
L8 :DO NI(8 )=NI(7 )+1 TO N;
    IF NC=9  THEN DO; CALL UP_DATE_H;
                      GO TO OUT8 ; END;
OUT8 : END L8 ;
OUT7 : END L7 ;
OUT6 : END L6 ;
OUT5 : END L5 ;
OUT4 : END L4 ;
OUT3 : END L3 ;
OUT2 : END L2 ;
OUT1 : END L1;
    END L0;
    PUT SKIP DATA(STEP(N),MC);
    DO I=1 TO MC-1; PUT EDIT(P(I))(F(5)); END;
    END;
END EXHAUST;
```